R & CDK: A Sturdy Platform in the Oceans of Chemical Data

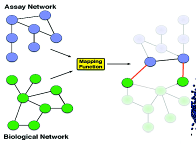
Rajarshi Guha

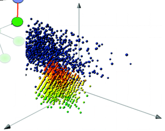
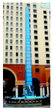
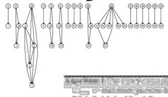
NIH Center for Translational Therapeutics

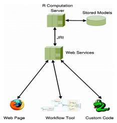
5th May, 2011
EBI, Hinxton

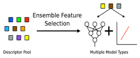
Background

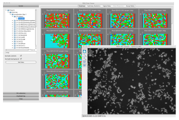
- Cheminformatics methods since 2003
  - QSAR, diversity analysis, virtual screening, fragments, polypharmacology, networks
- More recently
  - RNAi screening, high content screening
- Extensive use of machine learning
- All tied together with software development
  - User-facing GUI tools
  - Low level programmatic libraries
  - Core developer for the CDK
- Believer and practitioner of Open Source



[illegible]






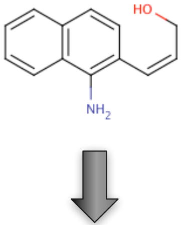
Why Cheminformatics in R?

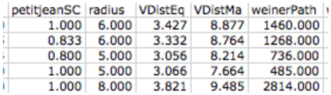
- In contrast to bioinformatics (cf. Bioconductor), not a whole lot of cheminformatics support for R
- For cheminformatics and chemistry relevant packages include
    - `rcdk`, `rpubchem`, `fingerprint`
    - `bio3d`, `ChemmineR`
- A lot of cheminformatics employs various forms of statistics and machine learning - R is exactly the environment for that
- We just need to add some chemistry capabilities to it

See here for a much more detailed tutorial on R & cheminformatics presented at the EBI in 2010

Motivations

- Much of cheminformatics is data modeling and mining
- But the numeric data is derived from chemical structure
- Thus we want to work with
  - molecules & and their parts
  - files containing molecules
  - databases of molecules





petitjeanSC	radius	VDistEq	VDistMa	weinerPath
1.000	6.000	3.427	8.877	1460.000
0.833	6.000	3.332	8.764	1268.000
0.800	5.000	3.056	8.214	736.000
1.000	5.000	3.066	7.664	485.000
1.000	8.000	3.821	9.485	2814.000

What is R?

- R is an environment for modeling
  - Contains many prepackaged statistical and mathematical functions
  - No need to implement anything
- R is a matrix programming language that is good for statistical computing
  - Full fledged, interpreted language
  - Well integrated with statistical functionality
  - More details later

What is R?

- It is possible to use R just for modeling
- Avoids programming, preferably use a GUI
  - Load data $\rightarrow$ build model $\rightarrow$ plot data
- But you can also get much more creative
  - Scripts to process multiple data files
  - Ensemble modeling using different types of models
  - Implement brand new algorithms
- R is good for prototyping algorithms
  - Interpreted, so immediate results
  - Good support for vectorization
    - Faster than explicit loops
    - Analogous to `map` in Python and Lisp
  - Most times, interpreted R is fine, but you can easily integrate C code

What is R?

- R integrates with other languages
  - C code can be linked to R and C can also call R functions
  - Java code can be called from R and vice versa. See various packages at rosuda.org
  - Python can be used in R and vice versa using Rpy
- R has excellent support for publication quality graphics
- See R Graph Gallery for an idea of the graphing capabilities
- But graphing in R does have a learning curve
- A variety of graphs can be generated
  - 2D plots - scatter, bar, pie, box, violin, parallel coordinate
  - 3D plots - OpenGL support is available

Parallel R

- R itself is not multi-threaded
  - Well suited for embarassingly parallel problems
- Even then, a number of “large data” problems are not tractable
  - Recent developments on integrating R and Hadoop address this
  - See the `RHIPE` package
- `snow` which allows distribution of processing on the same machine (multiple CPU’s) or multiple machines
- But see `snowfall` for a nice set of wrappers around `snow`
- Also see `multicore` for a package that focuses on parallel processing on multicore CPU’s

R and databases

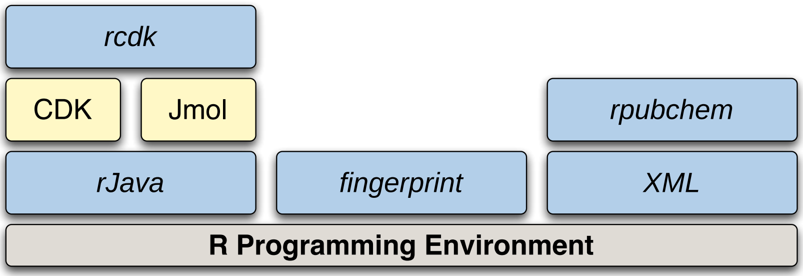
- Bindings to a variety of databases are available
  - Mainly RDBMS's but some NoSQL databases are being interfaced
- The R `DBI` spec lets you write code that is portable over databases
- Note that loading multiple database packages can lead to problems
- This can happen even when you don't explicitly load a database package
  - Some Bioconductor packages load the `RSQLite` package as a dependency, which can interfere with, say, `ROracle`

Why use a database?

- Dont have to load bulk CSV or .Rda files each time we start work
- Can index data in RDBMS's so queries can be very fast
- Good way to exchange data between applications (as opposed to .Rda files which are only useful between R users)

Using the CDK in R

- Based on the `rJava` package
- Two R packages to install (not counting the dependencies)
- Provides access to a variety of CDK classes and methods
- *Idiomatic R*



Acessibility & usability

- Plain R is not necessarily the most “usable” platform
- So `rcdk` doesn't really satisfy usability for complete R newbies
- But, if you know R, installation is trivial

```
> install.packages('rcdk', dependencies=TRUE)
```

- R specifies a documentation format
- Most packages have quite good documentation, `rcdk` is no exception
- A tutorial is also available from within R, in addition to the function docs

```
> vignette('rcdk') # read tutorial
> ls('package:rcdk') # list functions
> ?load.molecules # get help on a function
```

Reading in data

- The CDK supports a variety of file formats
- `rcdk` loads all recognized formats, automatically
- Data can be local or remote

```
mols <- load.molecules( c("data/io/set1.sdf",
              "data/io/set2.smi",
              "http://rguha.net/rcdk/remote.sdf"))
```

- Gives you a `list` of Java references representing `IAtomContainer` objects
- For large SDF's use an iterating reader
- Can't do much with these objects, except via `rcdk` functions

Writing molecules

- Currently only SDF is supported as an output file format
- By default a multi-molecule SDF will be written
- Properties are not written out as SD tags by default

```r
smis <- c("c1ccccc1", "CC(C=O)NCC", "CCCC")
mols <- sapply(smis, parse.smiles)

## all molecules in a single file
write.molecules(mols, filename="mols.sdf")

## ensure molecule data is written out
write.molecules(mols, filename="mols.sdf", write.props=TRUE)

## molecules in individual files
write.molecules(mols, filename="mols.sdf", together=FALSE)
```

Working with molecules

- Currently you can access atoms, bonds, get certain atom properties, 2D/3D coordinates
- Since `rcdk` doesn't cover the entire CDK API, you might need to drop down to the `rJava` level and make calls to the Java code by hand

Working with atoms

- Simple elemental analysis
- Identifying flat molecules

```
mol <- parse.smiles("c1ccccc1C(Cl)(Br)c1ccccc1")
atoms <- get.atoms(mol)

## elemental analysis
syms <- unlist(lapply(atoms, get.symbol))
round( table(syms)/sum(table(syms)) * 100, 2)

## is the molecule flat?
coords <- do.call("rbind", lapply(atoms, get.point3d))
any(apply(coords, 2, function(x) length(unique(x)) == 1))
```

SMARTS matching

- rcdk supports substructure searches with SMARTS or SMILES
- May not be practical for large collections of molecules due to memory

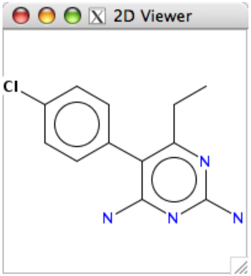
```
mols <- sapply(c("CC(C)(C)C",
                 "c1ccc(Cl)cc1C(=O)O",
                 "CCC(N)(N)CC"), parse.smiles)
query <- "[#6D2]"
hits <- matches(query, mols)

> print(hits)
CC(C)(C)C c1ccc(Cl)cc1C(=O)O CCC(N)(N)CC
FALSE TRUE TRUE
```

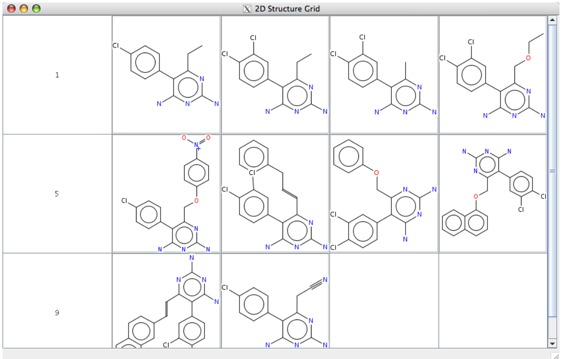
Visualization

- `rcdk` supports visualization of 2D structure images in two ways
- First, you can bring up a Swing window
- Second, you can obtain the depiction as a raster image
- **Doesn't work on OS X**

```r
mols <- load.molecules("data/dhfr_3d.sd")

## view a single molecule in a Swing window
view.molecule.2d(mols[[1]])

## view a table of molecules
view.molecule.2d(mols[1:10])
```

Visualization





Visualization

- The Swing window is a little heavy weight
- It'd be handy to be able to annotate plots with structures
- Or even just make a panel of images that could be saved to a PNG file
- We can make use of `rasterImage` and `rcdk`
- As with the Swing window, this won't work on OS X

```
m <- parse.smiles("c1ccccc1C(=O)NC")
img <- view.image.2d(m, 200,200)

## start a plot
plot(1:10, 1:10, pch=19)

## overlay the structure
rasterImage(img, 1,8, 3,10)
```

Molecular descriptors

- Numerical representations of chemical structure features
- Can be based on
  - connectivity
  - 3D coordnates
  - electronic properties
  - combination of the above
- *Many* descriptors are described and implemented in various forms
- The CDK implements 50 descriptor classes, resulting in $\approx$ 300 individual descriptor values for a given molecule

Descriptor caveats

- Not all descriptors are optimized for speed
- Some of the topological descriptors employ graph isomorphism which makes them slow on large molecules
- In general, to ensure that we end up with a rectangular descriptor matrix we do not catch exceptions
- Instead descriptor calculation failures return `NA`

CDK Descriptor Classes

- The CDK provides 3 packages for descriptor calculations
  - `org.openscience.cdk.qsar.descriptors.molecular`
  - `org.openscience.cdk.qsar.descriptors.atomic`
  - `org.openscience.cdk.qsar.descriptors.bond`
- `rcdk` only supports molecular descriptors
- Each descriptor is also described by an ontology
  - For `rcdk` this is used to classify descriptors into groups

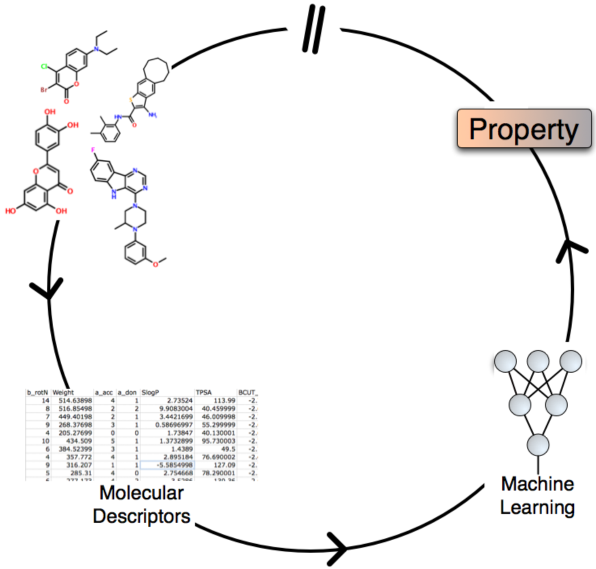
Descriptor calculations

- Can evaluate a single descriptor or all available descriptors
- If a descriptor cannot be calculated, `NA` is returned (so no exceptions thrown)

```
dnames <- get.desc.names('topological')
descs <- eval.desc(mols, dnames)
```

- Just evaluate topological descriptors
- descs will be a `data.frame` which can then be used in any of the modeling functions
- Column names are the descriptor names provided by the CDK

The QSAR workflow



b_rotN	Weight	a_acc	a_don	SlogP	TPSA	BCUT_
14	514.63898	4	1	2.73524	113.99	-2.
8	516.85498	2	2	9.9083004	40.459999	-2.
7	449.40198	2	1	3.4421699	46.009998	-2.
9	268.37698	3	1	0.58696997	55.299999	-2.
4	205.27699	0	0	1.73847	40.130001	-2.
10	434.509	5	1	1.3732899	95.730003	-2.
6	384.52399	3	1	1.4389	49.5	-2.
4	357.772	4	1	2.895184	76.690002	-2.
9	316.207	1	1	-5.5854998	127.09	-2.
5	285.31	4	0	2.754668	78.290001	-2.
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

The QSAR workflow

- Before model development you'll need to clean the molecules, evaluate descriptors, generate subsets
- With the numeric data in hand, we can proceed to modeling
- Before building predictive models, we'd probably explore the dataset
    - Normality of the dependent variable
    - Correlations between descriptors and dependent variable
    - Similarity of subsets
- Then we can go wild and build all the models that R supports

Accessing fingerprints

- CDK provides several fingerprints
  - Path-based, MACCS, E-State, PubChem
- Access them via `get.fingerprint(...)`
- Works on one molecule at a time, use `lapply` to process a list of molecules
- This method works with the `fingerprint` package
  - Separate package to represent and manipulate fingerprint data from various sources (CDK, BCI, MOE)
  - Uses C to perform similarity calculations
  - Lots of similarity and dissimilarity metrics available

Accessing fingerprints

```
mols <- load.molecules("data/dhfr_3d.sd")

## get a single fingerprint
fp <- get.fingerprint(mols[[1]], type="maccs")

## process a list of molecules
fplist <- lapply(mols, get.fingerprint, type="maccs")

## or read from file
fplist <- fp.read("data/fp/fp.data", size=1052,
                  lf=bci.lf, header=TRUE)
```

- Easy to support new line-oriented fingerprint formats by providing your own line parsing function (e.g., `bci.lf`)
- See the `fingerprint` package man pages for more details

Similarity metrics

- The `fingerprint` package implements 28 similarity and dissimilarity metrics
- All accessed via the `distance` function
- Implemented in C, but still, large similarity matrix calculations are not a good idea!

```
## similarity between 2 individual fingerprints
distance(fplist[[1]], fplist[[2]], method="tanimoto")
distance(fplist[[1]], fplist[[2]], method="mt")

## similarity matrix - compare similarity distributions
m1 <- fp.sim.matrix(fplist, "tanimoto")
m2 <- fp.sim.matrix(fplist, "carbo")

par(mfrow=c(1,2))
hist(m1, xlim=c(0,1))
hist(m2, xlim=c(0,1))
```

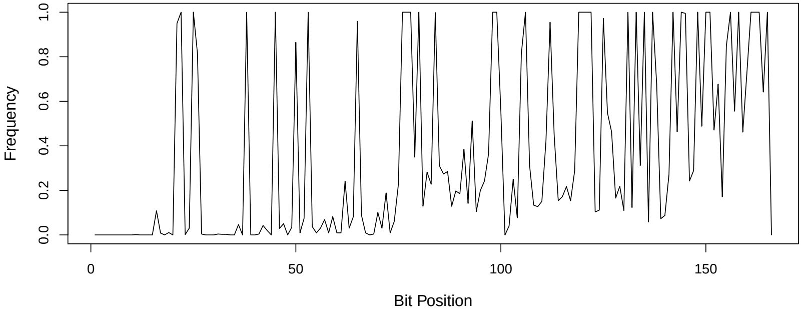
Comparing datasets with fingerprints

- We can compare datasets based on a fingerprints
- Rather than perform pairwise comparisons, we evaluate the normalized occurence of each bit, across the dataset
- Gives us a *n*-D vector - the “bit spectrum”

```
bitspec <- bit.spectrum(fplist)
plot(bitspec, type="l")
```

[0] Guha, R., *J. Comp. Aid. Molec. Des.*, **2008**, *22*, 367–384

Bit spectrum

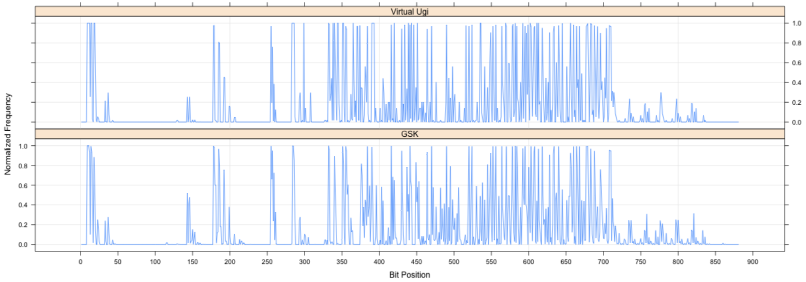


- Only makes sense with structural key type fingerprints
- Longer fingerprints give better resolution
- Comparing bit spectra, via any distance metric, allows us to compare datasets in $O(n)$ time, rather than $O(n^2)$ for a pairwise approach

[0] Guha, R., *J. Comp. Aid. Molec. Des.*, **2008**, *22*, 367–384

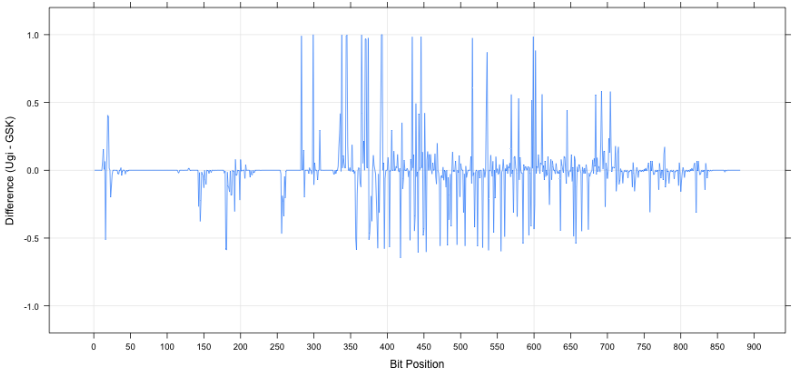
GSK & ONS Ugi datasets

- 117K member virtual library of Ugi products was the basis of an ONS project looking for anti-malarials (Jean-Claude Bradley, Drexel University)
- GSK recently published their anti-malarial screening dataset (13K compounds)
- How do the two data sets compare?



GSK & ONS Ugi datasets

- A little easier to identify differences if we take the “difference spectrum”



Comparing fingerprint performance

- Various studies comparing virtual screening methods
- Generally, metric of success is how many actives are retrieved in the top n% of the database
- Can be measured using ROC, enrichment factor, etc.
- Exercise - evaluate performance of CDK fingerprints using enrichment factors
  - Load active and decoy molecules
  - Evaluate fingerprints
  - For each active, evaluate similarity to all other molecules (active and inactive)
  - For each active, determine enrichment at a given percentage of the database screened

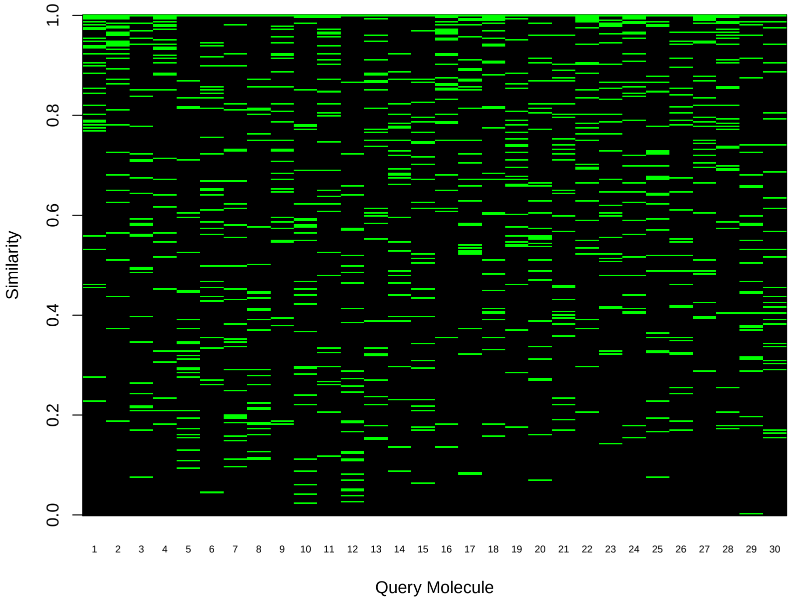
For a given query molecule, order dataset by decreasing similarity, look at the top 10% and determine fraction of actives in that top 10%

Comparing fingerprint performance

- A good dataset to test this out is the Maximum Unbiased Validation datasets by Rohr & Baumann
- Derived from 17 PubChem bioassay datasets and designed to avoid *analog bias* and *artifical enrichment*
- As a result, 2D fingerprints generally show poor performance on these datasets (by design)
- See here for a comparison of various fingerprints using two of these datasets

[0]Rohrer, S.G et al, *J. Chem. Inf. Model*, **2009**, *49*, 169–184
[0]Good, A. & Oprea, T., *J. Chem. Inf. Model*, **2008**, *22*, 169–178
[0]Verdonk, M.L. et al, *J. Chem. Inf. Model*, **2004**, *44*, 793–806

Comparing fingerprint performance



Fragment based analysis

- Fragment based analysis can be a useful alternative to clustering, especially for large datasets
- Useful for identifying interesting series
- Many fragmentation schemes are available
  - Exhaustive
  - Rings and ring assemblies
  - Murcko
- The CDK supports fragmentation (still needs work) into Murcko frameworks and ring systems

Getting fragments

- Access to exhaustive and Murcko fragmentation schemes
- Exhaustive fragmentation can take a long time in some cases
- Both have several parameters allow us to filter fragments

```
mol <- parse.smiles(
"c1cc(c(cc1c2c(nc(nc2CC)N)N)[N+](=O)[O-])NCc3ccc(cc3)C(=O)N4CCCCC4")

mfrags <- get.murcko.fragments(mol)
xfrags <- get.exhaustive.fragments(mol)
```

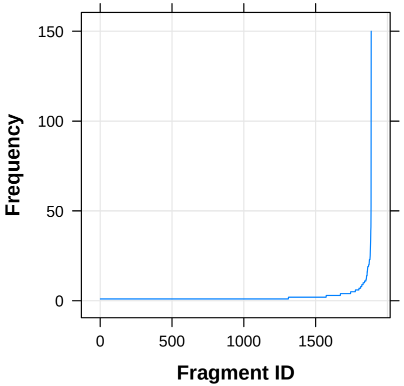
Doing stuff with fragments

- Look at frequency of occurence of fragments
- *Pseudo-cluster* a dataset based on fragments
- Compound selection based on fragment membership
- Develop predictive models on fragment members, looking for local SAR

Fragments & kinase activities

- Consider the Abbot kinase dataset (Metz et al)
- ≈ 1500 structures, 172 targets
- Slice and dice activities based on Murcko framework membership

```
frags <- lapply(mols,
   get.murcko.fragments)
fworks <-lapply(frags,
   function(x) x[[1]]$frameworks)
frag.freq <- data.frame(
   table(unlist(fworks))
)
```

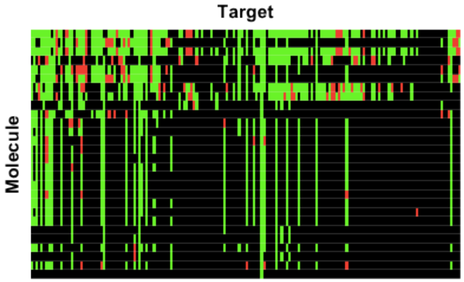


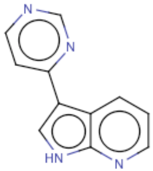
Fragments & kinase activities

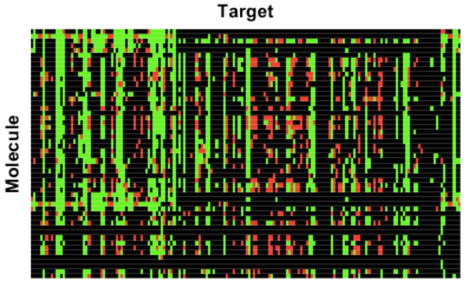
- Explore activity data on a fragment-wise basis
- Compare activity distributions by targets

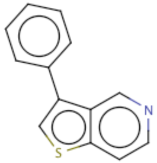
```
## build a look up table (frag SMILES -> molecule ID)
ftable <- do.call('rbind', mapply(function(x,y) {
  if (length(y) == 0) y <- NA
  data.frame(mid=x, frag=y)
}, names(fworks), fworks, SIMPLIFY=FALSE))
rownames(ftable) <- NULL
ftable <- subset(ftable, !is.na(frag))
ftable$frag <- as.character(ftable$frag)

## identify molecules containing a fragment
query <- 'c1nccc(n1)c3c[nH]c2ncccc23'
values <- subset(ftable, frag == query)
depvs <- subset(abbot, PUBCHEM_SID %in% values$mid)[, 15:186]
```

Fragments & kinase activities







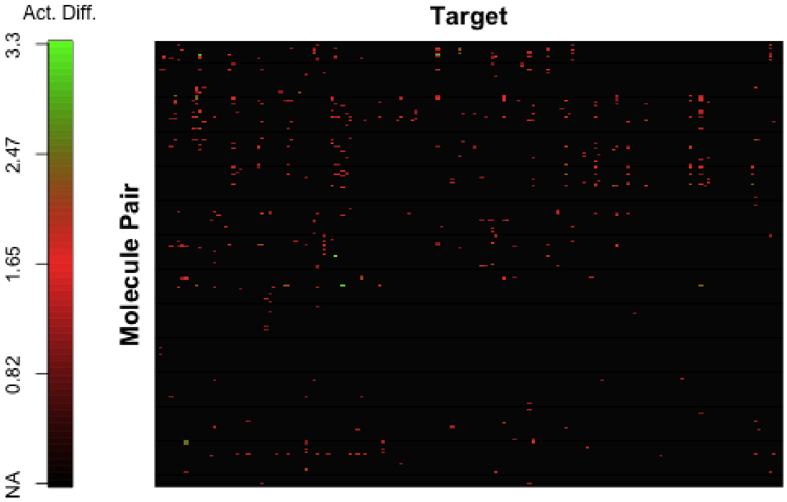


Matched molecular pairs

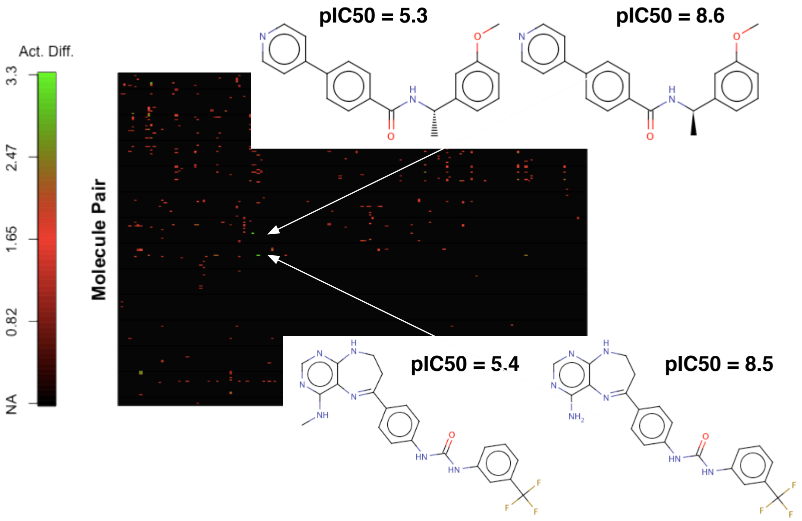
- Inspired by Gregs' 1-line SQL query
- But performed over 172 kinase targets
- Slower, especially the similarity matrix calculation

```r
## load molecules and get similarity matrix
mols <- load.molecules('abbot.smi')
fps <- fp.sim.matrix(lapply(mols, get.fingerprint, 'extended'))

## identify similar pairs
idxs <- which(fpsims > 0.95, arr.ind=TRUE)
idxs <- idxs[ idxs[,1] > idxs[,2], ] # ignore diagonal elements

## evaluate activity differences
mps <- t(apply(idxs, 1, function(x) {
  apply(depvs, 2, function(z) {
    d <- abs(z[x[1]] - z[x[2]])
    ifelse(d >= 1, d, NA)
  })
}))
```

Matched molecular pairs



Matched molecular pairs



Future developments

- One current drawback of the package is that most cheminformatics operations cannot be parallelized
  - Many objects are Java refs so can't be shared
  - Many CDK methods are not threadsafe
- Data table and depictions
- Streamline I/O and molecule configuration
- Add more atom and bond level operations
- Convert from `jobjRef` to S4 objects and vice versa
  - Would allow for serialization of CDK data classes
  - Is it worth the effort?

Acknowledgements

- `rcdk`
  - Miguel Rojas
  - Ranke Johannes
- CDK
  - Egon Willighagen
  - Christoph Steinbeck
  - ...